Study of different machine learning approaches for identification of fake news articles.

Josep Ricard Zurita Nicolas

July 26th 2019

.

Abstract

In this document multiple machine learning approaches, including Supervised, Semi-supervised and Unsupervised learning are explored with the objective of finding the best algorithm for the task of identifying fake news. The corpus used consists on pure text data extracted from news articles. TF-IDF and word2vec features are studied. Python is used for the implementation.

.

アブストラクト

この文書の目的は「教師あり学習」,「教師なし学習」,「半教師付き学習」の様々な手法を用いてフェイクニュースの指摘に最も相応しいアルゴリズムを見出すことです。データとしてはニュース記事のテキストのみが使われています。TF-IDF と word2vec のフィーチャーが検討されます。パイソンで実現されています。

.

Abstracto

En este documento se exploran múltiples métodos de machine learning, incluyendo aprendizaje supervisado, semisupervisado y no supervisado, con el objetivo de encontrar el algoritmo más apropiado para la tarea de identificar noticias falsas. El corpus utilizado está compuesto únicamente texto extraído de artículos de noticias. Se han estudiado las features TF-IDF y word2vec. El proyecto se ha implementado en Python.

.

Resum

En aquest document s'exploren diferents mètodes de Machine Learning, incloent aprenentatge supervisat, semi supervisat i no supervisat, amb l'objectiu de trobar l'algoritme més apropiat per a la tasca d'identificar noticies falses. S'ha fet servir un corpus compost únicament de text extret d'articles de noticies. S'han estudiat les features TF-IDF i word2vec. El projecte s'ha implementat en Python.

1 Introduction

Recently, the world has seen a seen a series of technological and social changes that have ultimately affected real life events like the most recent USA presidential elections. News articles are no longer written and shared with the purpose of delivering reliable information to the public, but are sometimes created with the intent of spreading false information or simply for economic gain.

This project has studied multiple Machine Learning approaches to classify full news articles as "Real News" or "Fake News", and compares the success of each approach.

The considered approaches have been the classic supervised learning algorithms, some semi-supervised approaches and finally completely unsupervised learning approaches for clustering. First, these approaches were tested with TF-IDF features. Afterwards, some of the same approaches were tested with word2vec and doc2vec features.

The objective of the project has been to find the best configuration of features and algorithm that produces the best results in terms of successful classification of the news articles.

The programming was implemented in the platform Google Colaboratory, using Python as the language.

2 Datasets

For this project, the raw data consisted of full news article's texts. The datasets used have been "Getting Real about Fake News" [4] and "All the news" [6], both of which are hosted in the dataset sharing website Kaggle.

Dataset [4] consists of 13000 news articles collected from 244 unreliable news sites, complete with title, content and other information. Dataset [6] consists of approximately 143000 news articles from at least somewhat respectable news sites. Some of the articles may be biased towards a certain political view but they are nevertheless more true than false. The dataset includes title and content of the articles, along with other information.

As the datasets are very different in size, only 10000 articles from each set were used for training purposes, and another 2000 for testing. These articles were selected at random every time the code was executed to guarantee that the implementation is reliable regardless of the selected articles.

From both datasets only the raw text of the article has been used. A 95% of these articles are written in English, while the rest are written in Spanish, Russian, Arabic and other languages. The preprocessing deletes all non-alphanumeric characters, so non-alphabet articles would be reduced to blanks (or numbers only if present). In some other cases, like Spanish, the words will remain but will not provide valuable information, as the stemming will not be adequate and the words will be uncommon enough to be left out of feature vectors. These articles could have been filtered but were left in as a sort of noise. Non-alphabet languages make up about half of the noise articles, while the rest are non-English alphabet languages.

3 TF-IDF features

3.1 Preprocessing and feature extraction

The Python libary "sklearn" [2] greatly simplified the pre-processing of the raw text, as it has functions that take raw text, perform a series of operations with it and return a vocabulary and a matrix where every row is a vector with the counted words in the article.

The performed operations are transformation into lower case, removal of symbols, tokenization, removal of stop words (common words) and finally counting of words. An example of this process can be seen in Figure 1.

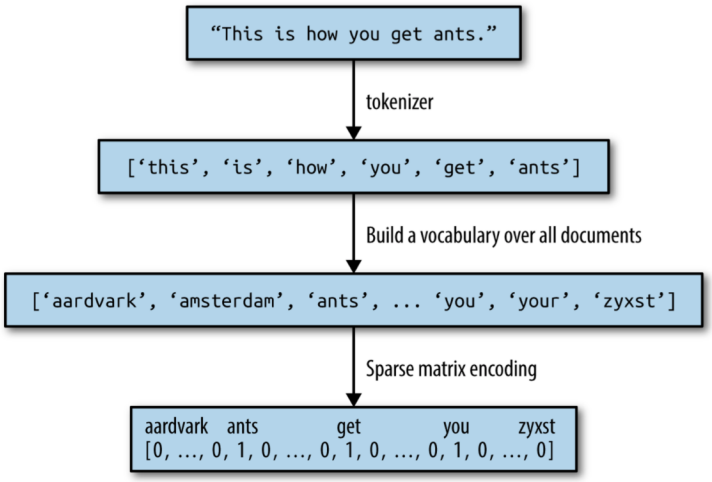

Figure 1: Example of text processing.

With a small modification of the function, the operation of stemming was also added to the pipeline. This reduces words to their stem form to produce better results when classifying.

Note that different tokenizers and stemmers were used and the results did not present a big enough difference to justify adding the results from each one.

With the resulting "Bag of Words", another function was used to obtain a similar matrix but with a TF-IDF value instead, which represents the relevance of a word in a document referenced to the relevance of the word in all documents.

This matrix has a shape of MxN, where M is the number of articles used for training, 20000, and N is the number of features or simply the N most common words along the M articles in the collection. Different values for N have been tested to see what amount of features produces the best results.

3.2 Supervised learning results

3.2.1 Feature size selection

First, an appropriate number of features (N) was selected. To do this the results from multiple algorithms with default configurations were compared. These algorithms are K-Nearest Neighbors (KNN, K=5), Support Vector Machine (SVM, RBF kernel and C=1), Multinomial Naive Bayes (MNB, alpha=1) and Decision Trees (DT, Gini criterion). More in depth default parameters can be found at each model's documentation [2].

Table 1: Resulting f-scores using different features sizes and default classifiers.

N	**100**	**300**	**500**	**1000**	**3000**	**5000**
KNN	0.7660	0.7158	0.6420	0.5181	0.3541	0.3064
SVM	0.8339	0.8649	0.8826	0.9019	0.9121	0.9175
MNB	0.7708	0.7979	0.8059	0.8220	0.8361	0.8453
DT	0.7371	0.7665	0.7730	0.7927	0.7988	0.8099

As can be seen in Table 1, the accuracy of the KNN classifier, in this case with K=1, decreased when the amount of features was greater than 100. With the rest of classifiers, the accuracy increased as more features were used.

Since execution time was not relevant for this implementation, as it is not supposed to be used in real time, 5000 features were used on on all algorithms except KNN, where only 100 features were used.

3.2.2 K-Nearest Neighbors

Table 2: F-scores of KNN algorithm for different values of K.

K	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**
%	0.7441	0.7597	0.7660	0.7688	0.7702	0.7703	0.7678	0.7709

In Table 2 we can see that increasing the value of K also increased the f-score of the classifier, and thus best value was K=15 with an f-score of 0.7709.

3.2.3 Support Vector Machine

Table 3: F-score of SVM algorithm for different kernel types and values of C.

C	**0.1**	**0.2.**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
RBF	0.8705	0.8801	0.8906	0.8986	0.9036	0.9065	0.9092	0.9111	0.9124	0.9133
LIN	0.8983	0.9074	0.9100	0.9118	0.9100	0.9117	0.9124	0.9109	0.9115	0.9107
POLY	0.7300	0.7774	0.8112	0.8373	0.8544	0.8683	0.8849	0.8898	0.8933	0.8946
SIG	0.8837	0.8941	0.8968	0.8989	0.8997	0.9014	0.9021	0.9005	0.9003	0.9001

In Table 3 we can see the f-score of the SVM using four different kernels, namely the Radial Basis Function kernel, a linear kernel, a third degree polynomial kernel and a sigmoid kernel. All of these kernels were tested with values of C ranging from 0.1 to 1.

It can be appreciated that the best f-score, 0.9133, was obtained with the RBF kernel and C=1.

3.2.4 Multinomial Naive Bayes

Table 4: F-score of the Multinomial Naive Bayes algorithm with different values for the smoothing parameter alpha.

alpha	**0**	**0.1**	**0.2.**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Accuracy	0.8495	0.8475	0.8472	0.8467	0.8462	0.8460	0.8457	0.8455	0.8453	0.8455	0.8453

A Multinomial version of Naive Bayes was used because the features extracted from the text data consisted of a sparse matrix of great dimensions.

The results of the Multinomial Naive Bayes can be seen in Table 4 where the algorithm was tested for multiple values of the smoothing parameter. It can be observed that the algorithm obtained the highest score when alpha=0, although the scores did not vary significantly.

3.2.5 Decision Trees

Table 5: F-score of the Decision Trees algorithm using both possible criterions.

Criterion	**Gini**	**Entropy**
Accuracy	0.8076	0.7989

Finally the last algorithm studied was the Decision Trees algorithm, where both measures of quality of a split were considered. The default parameters for limiting the size of the tree were used, which means that no nodes were pruned. The Google Colaboratory platform offers enough memory to allow the tree to fully grow.

We can see in Table 5 that the Gini criterion produces slightly better results. Regardless, the scores are much lower than those of previous algorithms.

3.3 Unsupervised learning results

The same data and preprocessing were used for this section, the algorithms were simply not informed of the known labels of the data. Instead, these labels have been used to check the quality of the classification. Note that Unsupervised learning algorithms return the data labeled as [0,1], which may or may not correspond to the real labels. It has been assumed that the highest F-score is obtained when the labels match.

3.3.1 K-Means Clustering

The first clustering algorithm tested is the simple iterative K-Means algorithm, which was considered adequate to deal with a 5000 dimensional problem and give an idea of the complexity of the classification.

In Table 6 we can see that the trained classifiers tends to classify most data as fake, and can only reach an F-measure value of around 0.61. Both Training and Testing data results are compared, and it is clear that the classifier is consistent regardless of the data provided.

Table 6: Confusion matrix and F-measure results for the K-means algorithm.

Training data results

N = 20000	**Predicted Fake**	**Predicted True**		**Precision**	**Recall**	**F-measure**
Actual Fake	**7606**	2394	10000	0.5496	0.2922	0.3815
Actual True	7078	**2922**	10000	0.5179	0.7606	**0.6162**
	14684	5316				

Testing data results

N = 4000	**Predicted Fake**	**Predicted True**		**Precision**	**Recall**	**F-measure**
Actual Fake	**1521**	479	2000	0.5531	0.2965	0.3860
Actual True	1407	**593**	2000	0.5194	0.7605	**0.6172**
	2928	1072				

3.3.2 Birch Clustering

Next is the Birch algorithm, which should also be able an acceptable choice for such a big feature selection.

We can see in Table 7 that the results are actually worse than with K-Means, as the maximum F-score is 0.59. In this Training data it can also be appreciated that the algorithm tends to classify more Fakes than Reals.

But there is a very noticeable difference, and it is the inconsistency of the results in the Testing data. Using the same labels for both tables, the Testing data produces the direct opposite effect, classifying most data as True, and obtaining a very low F-score. This can only be explained by assuming that the classifier has somehow become over-fitted for two clusters that are wrong in the first place.

3.3.3 Agglomerative Clustering

The last algorithm studied is the Agglomerative Clustering algorithm, which simply joins together the samples and clusters that are closest in distance. This should make it a decent choice for working with many samples and features.

As this algorithm only creates clusters with the original data, the results in Table 8 are those of the Training data only.

In this case we can also appreciate that the algorithm considers most data as Fake, and only manages an F-score of around 0.6.

Table 7: Confusion matrix and F-measure results for the Birch algorithm.

Training data results

N = 20000	Predicted Fake	Predicted True	
Actual Fake	**6576**	3424	10000
Actual True	5482	**4518**	10000
	12058	7942	

Precision	Recall	F-measure
0.5688	0.4518	0.5036
0.5453	0.6576	**0.5962**

Testing data results

N = 4000	Predicted Fake	Predicted True	
Actual Fake	**575**	1425	2000
Actual True	605	**1395**	2000
	1180	2820	

Precision	Recall	F-measure
0.4946	0.6975	0.5788
0.4872	0.2875	**0.3616**

Table 8: Confusion matrix and F-measure results for Agglomerative Clustering.

Training data results

N = 20000	Predicted Fake	Predicted True	
Actual Fake	**8074**	1926	10000
Actual True	8530	**1470**	10000
	16604	3396	

Precision	Recall	F-measure
0.4862	0.8074	0.6069
0.4328	0.1470	0.2194

3.4 Semi-supervised learning results

To test this approach only the Label Propagation algorithm is studied. This algorithm works by giving each point the label that is most common among its close neighbors, which means that it can work with any amount of labeled data.

In Table 9 we can see the results of testing the algorithm with different ratios of labeled data and the two possible functions to determine distance between data-points.

The RBF and KNN Kernels were configured with the parameters that produced the best results in the Supervised Learning part of the study.

It can be appreciated that the results with only a 20% of labeled data are already significantly better than those obtained from Unsupervised Learning, and also it can be seen that increasing the labeled data above 50% does not increase the scores.

The full confusion matrix of the 50% labeled data and RBF Kernel can be seen in Table 10. The algorithm does not appear to lean towards any specific prediction and produces consistent results.

Table 9: Results of the Label Propagation algorithm with different amounts of labeled data and kernels, expressed as F-measure.

N = 4000	20% Labeled	50% Labeled	80% Labeled
RBF Kernel	0.7148	0.8243	0.8231
KNN Kernel	0.6782	0.7007	0.7170

Table 10: Detailed confusion matrix of the best Semi-supervised results.

N = 4000	Predicted Fake	Predicted True	
Actual Fake	**1673**	327	2000
Actual True	386	**1614**	2000
	2059	1941	

Precision	Recall	F-measure
0.8125	0.8365	**0.8243**

3.5 TF-IDF Conclusions

From the results of the Supervised learning section we can observe that the algorithm best suited for the task of classifying fake news is the Support Vector Machine algorithm, by a significant margin.

It is worth mentioning that this algorithm is much slower than the rest, taking up to 15 minutes to train and test the classifier once. Other classifiers, like Naive Bayes, required no longer than 5 seconds. As this is not a real time implementation we can compromise speed in favor of the increased accuracy SVM offers.

The results from pure Unsupervised Learning are mediocre at best, with an average F-score of 0.6 it can be concluded that, as expected, the clustering problem is far too complex to be successfully performed without any labeled data.

It can be assumed that the articles are being classified into undesired clusters, like [politics, rest] or [right wing, left wing].

It is possible that by fine-tuning the parameters in these algorithms the results improved, but it is not reasonable to expect a major improvement from the Unsupervised Learning approach and it can be discarded as a viable solution for this problem.

On the other hand, the Semi-supervised approach has produced quite acceptable results, with scores around 0.7 with only 20% of labeled data and above 0.8 with 50% of labeled data. These results are still below the performance of some of the Supervised Learning algorithms, but there is merit in obtaining such results with only half of the labels.

4 Word2vec and doc2vec features

Word2vec is a word embedding algorithm similar in concept to word count or TF-IDF. The algorithm looks at a certain word and any neighboring words around it, within a window of size K. The window will slide along all words in the phrase or text, as can be seen in Fig. 2, and will create pairs of words that should be related.

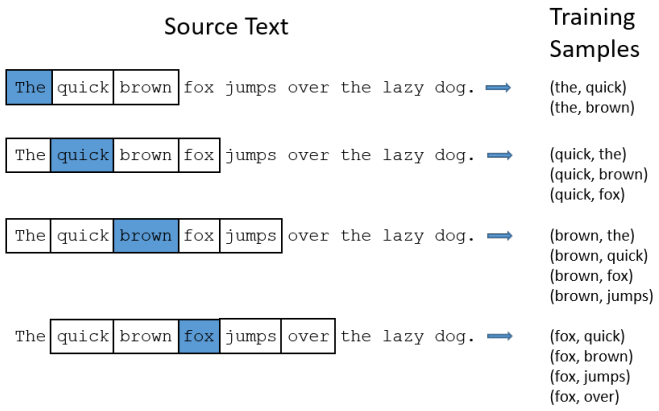

Figure 2: Sliding window of size K=2 [5].

After this procedure is done along the entire text, each word in the vocabulary is represented by a one-hot vector, which will be used as input for a Deep Neural Network, while the words paired with it previously will be the desired outputs.

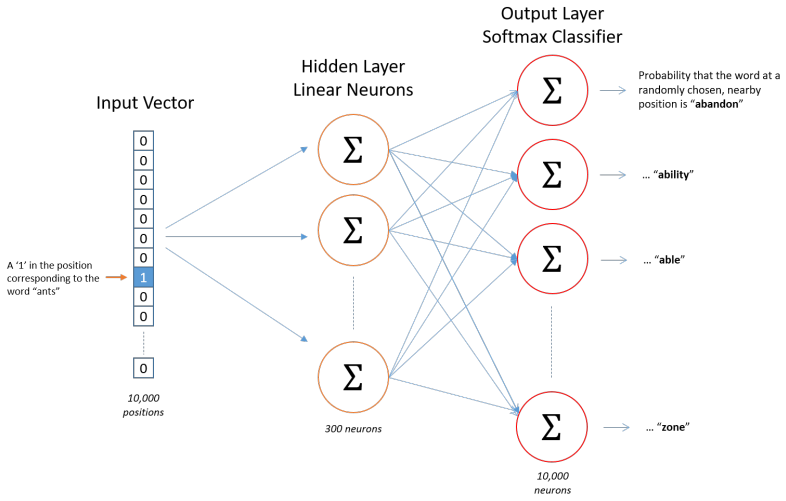

Figure 3: Structure of the Neural Network [5].

In Fig. 3 we can see an example of the structure of such a Neural Network, where the input and output layers are of size 10000 and the hidden layer is of length 300. For each input all possible pairs will be used as desired output and the Network will be trained by Back-Propagation.

After the Network is trained for each input, the weights of the hidden layer will be saved as a vector of length 300. This vector describes the relationship between the input word and all other words in the vocabulary.

The doc2vec features are much simpler and simply generate a vector that represents the entire text, after being trained with a full corpus of multiple texts.

Both word2vec and doc2vec are implemented using the library gensim[3].

4.1 Preprocessing and feature extraction

As the library Scikit-Learn[2] can no longer be used for feature extraction, as it was done at the same time as the TF-IDF feature extraction, the preprocessing now has to be done separately with the Natural Language Toolkit for Python[1] (NLTK). The text is preprocessed as similarly as possible to the preprocessing performed previously, with the exception of stemming which is no longer performed, as the word2vec features should be robust enough to deal with multiple words with similar meaning.

A word2vec (skip-gram) model is trained with the entire corpus, which results in a vocabulary of words, each with a vector associated. The length of the vector was changed during the experiments.

Each of the news articles is described by a vector which is the average vector of all the words it is composed of. Put simply, the text [dog jumps fence] would be described by adding the vectors for [dog], [jumps] and [fence] and dividing the resulting vector by 3 (the amount of words).

Doc2vec features do not require such treatment as they generate such a vector automatically.

4.2 Models

The experiments consisted in comparing the results obtained by different models or sets of features, which are described as follows:

- Model A: Skip-Gram word2vec, vector size = 100, window size = 2.
- Model B: Skip-Gram word2vec, vector size = 300, window size = 2.
- Model C: Skip-Gram word2vec, vector size = 100, window size = 3.
- Model D: Skip-Gram word2vec, vector size = 300, window size = 3.
- Model E: Google's word2vec model.
- Model F: Doc2vec, vector size = 100, window size = 2.
- Model G: Doc2vec, vector size = 300, window size = 2.
- Model H: Doc2vec, vector size = 100, window size = 3.
- Model I: Doc2vec, vector size = 300, window size = 3.

Note that the variables are the length of the feature vector and the size of the window. Google's own trained word2vec model is also included for reference, as Model E. It contains about 3 million words trained from 100 billion words from Google News. Its feature vector size is 300.

4.3 Results

The algorithms chosen to test the word2vec and doc2vec features are those that produced the best results in the previous section, as both types of data have a very similar shape and characteristics.

The f-scores obtained by using the 9 different models with different algorithms are below.

Table 11 includes the results from the K-Nearest Neighbors algorithm, with different K values.

Tables 12, 13, 14 and 15 detail the results from the SVM algorithm using different kernels each. The C variable is explored with each kernel.

The Multinomial Naive Bayes algorithm is in Table 16.

Finally, Table 17 summarizes the semi-supervised Label Propagation algorithm, with different ratios of labeled data. Note that this last table does not have results for the doc2vec features, as these features did not manage to appropriately train the model, meaning that regardless of parameters and iterations, the training converged into a model that classiffied over 99% of the samples as 'True'.

Table 11: F-scores obtained from the KNN algorithm.

K	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**
Model A	0.8024	0.8115	0.8160	0.8138	0.8138	0.8095	0.8101	0.8131
Model B	0.8163	0.8228	0.8200	0.8207	0.8193	0.8178	0.8163	0.8120
Model C	0.8166	0.8271	0.8281	0.8268	0.8258	0.8248	0.8196	0.8219
Model D	0.8191	0.8283	0.8253	0.8273	0.8242	0.8205	0.8200	0.8193
Model E	0.8035	0.8098	0.8106	0.8119	0.8097	0.8083	0.8072	0.8047
Model F	0.7121	0.7202	0.7185	0.7161	0.7191	0.7152	0.7158	0.7143
Model G	0.7159	0.7329	0.7231	0.7175	0.7156	0.7155	0.7086	0.7063
Model H	0.7104	0.7155	0.7006	0.7024	0.7086	0.7051	0.6967	0.6908
Model I	0.7174	0.7144	0.7106	0.7046	0.6991	0.7023	0.6953	0.6900

Table 12: F-scores obtained from the SVM algorithm using an RBF kernel.

C (RBF Kernel)	**0.1**	**0.3**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
Model A	0.8027	0.8208	0.8234	0.8245	0.8251	0.8256	0.8260	0.8259
Model B	0.8130	0.8306	0.8380	0.8402	0.8409	0.8444	0.8478	0.8512
Model C	0.8054	0.8209	0.8275	0.8269	0.8269	0.8283	0.8289	0.8284
Model D	0.8222	0.8384	0.8453	0.8486	0.8506	0.8528	0.8523	0.8524
Model E	0.8154	0.8349	0.8429	0.8445	0.8448	0.8466	0.8489	0.8489
Model F	0.7141	0.7323	0.7387	0.7416	0.7430	0.7436	0.7434	0.7447
Model G	0.7094	0.7372	0.7442	0.7473	0.7493	0.7498	0.7509	0.7505
Model H	0.6834	0.7141	0.7255	0.7269	0.7295	0.7286	0.7311	0.7332
Model I	0.6837	0.7178	0.7283	0.7269	0.7302	0.7327	0.7363	0.7381

Table 13: F-scores obtained from the SVM algorithm using a linear kernel.

C (LIN Kernel)	0.1	0.3	0.5	0.6	0.7	0.8	0.9	1
Model A	0.8084	0.8112	0.8138	0.8139	0.8127	0.8138	0.8140	0.8156
Model B	0.8259	0.8342	0.8384	0.8383	0.8382	0.8382	0.8407	0.8388
Model C	0.8087	0.8167	0.8196	0.8198	0.8195	0.8191	0.8205	0.8214
Model D	0.8319	0.8401	0.8427	0.8414	0.8420	0.8417	0.8431	0.8433
Model E	0.7913	0.8109	0.8145	0.8158	0.8158	0.8157	0.8161	0.8167
Model F	0.6354	0.6339	0.6342	0.6333	0.6332	0.6332	0.6340	0.6334
Model G	0.6526	0.6511	0.6504	0.6498	0.6488	0.6480	0.6485	0.6478
Model H	0.6029	0.5964	0.5938	0.5935	0.5931	0.5941	0.5939	0.5939
Model I	0.6174	0.6092	0.6044	0.6079	0.6108	0.6117	0.6122	0.6107

Table 14: F-scores obtained from the SVM algorithm using a Polynomial kernel.

C (POLY Kernel)	0.1	0.3	0.5	0.6	0.7	0.8	0.9	1
Model A	0.7997	0.8144	0.8202	0.8226	0.8236	0.8236	0.8250	0.8265
Model B	0.8148	0.8338	0.8392	0.8406	0.8416	0.8442	0.8461	0.8472
Model C	0.8059	0.8215	0.8250	0.8265	0.8282	0.8272	0.8278	0.8291
Model D	0.8247	0.8386	0.8421	0.8452	0.8458	0.8478	0.8497	0.8503
Model E	0.8176	0.8363	0.8426	0.8435	0.8465	0.8487	0.8509	0.8525
Model F	0.7054	0.6903	0.6864	0.6849	0.6858	0.6862	0.6868	0.6879
Model G	0.7078	0.6879	0.6861	0.6840	0.6823	0.6829	0.6830	0.6816
Model H	0.6393	0.6105	0.6062	0.6055	0.6075	0.6090	0.6092	0.6085
Model I	0.6237	0.6106	0.6084	0.6085	0.6080	0.6112	0.6123	0.6126

Table 15: F-scores obtained from the SVM algorithm using a Sigmoid kernel.

C (SIG Kernel)	0.1	0.3	0.5	0.6	0.7	0.8	0.9	1
Model A	0.7534	0.6721	0.6421	0.6348	0.6309	0.6296	0.6269	0.6242
Model B	0.7707	0.7119	0.6807	0.6708	0.6643	0.6597	0.6574	0.6545
Model C	0.7767	0.7288	0.6939	0.6843	0.6816	0.6749	0.6715	0.6678
Model D	0.7860	0.7467	0.7117	0.7013	0.6946	0.6921	0.6857	0.6827
Model E	0.7032	0.6788	0.6686	0.6635	0.6624	0.6579	0.6578	0.6576
Model F	0.4032	0.4314	0.4352	0.4351	0.4347	0.4346	0.4363	0.4374
Model G	0.4286	0.4270	0.4328	0.4324	0.4351	0.4350	0.4336	0.4332
Model H	0.4067	0.4281	0.4314	0.4319	0.4316	0.4327	0.4335	0.4331
Model I	0.4053	0.4172	0.4184	0.4199	0.4189	0.4197	0.4182	0.4182

Table 16: F-scores obtained from the Multinomial Naive Bayes algorithm with different smoothing parameters.

alpha	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Model A	0.7684	0.7684	0.7684	0.7684	0.7682	0.7682	0.7682	0.7682	0.7682	0.7682
Model B	0.7415	0.7415	0.7415	0.7415	0.7415	0.7415	0.7415	0.7415	0.7415	0.7415
Model C	0.7485	0.7485	0.7485	0.7485	0.7485	0.7485	0.7485	0.7485	0.7485	0.7485
Model D	0.7523	0.7523	0.7523	0.7523	0.7523	0.7523	0.7523	0.7523	0.7523	0.7523
Model E	0.6951	0.6951	0.6951	0.6951	0.6951	0.6951	0.6951	0.6951	0.6951	0.6951
Model F	0.5897	0.5897	0.5897	0.5897	0.5897	0.5897	0.5897	0.5897	0.5897	0.5897
Model G	0.6082	0.6082	0.6082	0.6082	0.6082	0.6082	0.6082	0.6082	0.6082	0.6082
Model H	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861
Model I	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861	0.5861

Table 17: F-scores obtained from the Label Propagation algorithm (RBF kernel), using different amounts of labeled data.

N=4000	10% Labeled	20% Labeled	30% Labeled	50% Labeled	80% Labeled
Model A	0.7557	0.7701	0.7824	0.8029	0.8222
Model B	0.7327	0.7793	0.7972	0.8080	0.8278
Model C	0.7693	0.7939	0.8012	0.8115	0.8264
Model D	0.7410	0.7831	0.7986	0.8109	0.8305
Model E	0.7437	0.7557	0.7768	0.7797	0.7868

4.4 Word2vec and Doc2vec Discussion and Conclusions

From the results, the first thing to observe is that doc2vec features are much worse than word2vec features. The f-scores are much lower in every algorithm and for Semi-supervised learning they are not able to train a model.

Then we can observe that the performance of the Google pre-trained word2vec model is slightly lower than that of the locally trained model. This seems to indicate that if a big enough dataset is available, it's better to train a model than to rely on Google's.

Finally we can observe a pattern where the bigger window size and longer vector size are in general the best option. Increasing the vector size seems to net better results than increasing the window size, but it's best to do both if possible.

For the KNN algorithm, Models C and D performed the best, with values of K equal to 5 and 3 respectively.

In SVM algorithms Model D was the best overall but was out-shined by Google's Model E in the polynomial kernel version.

In MNB, Model A performed the best although the results are low in general. Note that changing the alpha value does not affect the results.

Lastly, in Label Propagation we can see Model C perform the best for low amounts of labeled data, while Model D is the best for high amounts of labeled data.

Finally we can say that the configurations that produced the best results are obtained with SVM (RBF kernel) using Model D's features.

5 Additional comparison of TF-IDF and word2vec

After seeing the results from TF-IDF and word2vec and comparing them, it can be appreciated that TF-IDF manages to get scores over 0.9 while word2vec stalls at around 0.85.

Both features should contain a similar amount of information, so the main difference is the length of the feature vectors. I already proved with my first experiments that TF-IDF performs better as the feature vector size increases, so I considered it worthwhile to extend the feature vector size of the word2vec features and compare the results.

The algorithms used for this experiments are three different types of SVM, with a reduced amount of values of C. These algorithms are chosen because they performed the best with both sets of features.

Tables 18 and 19 represent the f-scores obtained with feature vectors of sizes 1000 and 3000 respectively. The window size is always 3. Originally I intended to obtain results for vector sizes up to 5000, but the limitations of the platform Google Colaboratory made it restart every time a word2vec model was trained with such a vector size.

Regardless, it can be appreciated that increasing the vector size to 1000 increases the scores slightly over size 300, but increasing them further does not increase the scores and actually reduces them in some cases.

Therefore, even though this application does not require fast execution times, there is no reason to increase the vector sizes over 1000.

Table 18: Results from vector size 1000 word2vec features with different SVM kernels.

s = 1000	**0.1**	**0.7**	**0.8**	**0.9**	**1**
RBF	0.8142	0.8539	0.8565	0.8589	0.8617
LIN	0.8266	0.8556	0.8569	0.8573	0.8584
POLY	0.8132	0.8586	0.8595	0.8616	0.8638

Table 19: Results from vector size 3000 word2vec features with different SVM kernels.

s = 3000	**0.1**	**0.7**	**0.8**	**0.9**	**1**
RBF	0.8215	0.8590	0.8597	0.8621	0.8633
LIN	0.8378	0.8527	0.8530	0.8557	0.8558
POLY	0.8197	0.8602	0.8614	0.8623	0.8630

6 Final Conclusions

In terms of Supervised Learning it can be concluded that the SVM algorithms are superior in every aspect to all other alternatives. The scores reached by SVM and TF-IDF features are unmatched by any other configuration and should always be considered at least as a decent option when tackling problems regarding the classification of news articles.

Unsupervised Learning or Clustering, on the other hand, should immediately be discarded as an option when dealing with the fake news problem. The task is far too

complex to expect the algorithms to pick on the subtleties of 'fake' and 'real' without any guidance.

Finally, if the available labeled data is of reduced quantity, a Semi-supervised approach can be viable. In these cases the word2vec features offer a better performance at 10% labeled data than TF-IDF can offer at 20% labeled data. Suprisingly, the results for a low amount of labeled data are better when the vector size is smaller.

Further studies about different preprocessing techniques, optimal parametrization of the algorithms and usage of different features derived from word2vec and doc2vec may serve to reach a deeper understanding of the topic.

References

[1] Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python.* O'Reilly Media Inc., 2009.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[4] Megan Risdal. Getting real about fake news. https://www.kaggle.com/mrisdal/fake-news. Accessed: 2019-05-15.

[5] Gonzalo Ruiz de Villa. Introducción a word2vec (skip gram model). https://medium.com/@gruizdevilla/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f. Accessed: 2019-06-20.

[6] Andrew Thompson. All the news. https://www.kaggle.com/snapcrack/all-the-news. Accessed: 2019-05-15.